MyGa : manual de utilização

Adel BEN HAJ YEDDER.

19 de Novembro de 2002

Tradução do francês para o português por:

Dr. Adriano Pinto Mariano
Dra. Caliane Bastos Borba Costa

Laboratório de Otimização, Projeto e Controle Avançado (LOPCA)
Responsável: Prof. Dr. Rubens Maciel Filho
Faculdade de Engenharia Química (FEQ)
Universidade Estadual de Campinas (UNICAMP) (Brasil)

01 de Julho de 2008

1. Introdução

Este programa é uma implementação de um algoritmo genético em código real escrito em Fortran. Os tipos de operadores desenvolvidos são bastante limitados e somente alguns operadores conhecidos por sua eficiência foram incorporados. O objetivo desse programa, quando foi desenvolvido, era de ter um algoritmo genético em Fortran de fácil utilização e eficiente para o problema para o qual ele foi escrito.

O programa é distribuído a fim de fornecer aos programadores em Fortran uma ferramenta para testar e/ou utilizar uma otimização com algoritmos genéticos em seus próprios problemas.

Este programa é fornecido sem qualquer garantia nem qualquer suporte além desse documento. Se você o utiliza, você é o ÚNICO responsável por sua utilização e em particular pelos resultados que ele gere e por aquilo que você faz em seguida.

2. Instalação

Sistema operacional: O programa foi escrito e testado para ***Linux***. Entretanto, deve funcionar em outros sistemas do tipo *N*X dependendo de pequenas modificações.
Compilador: O programa está escrito em ***Fortran 77***. Você deve então utilizar um compilador compatível com o Fortran 77. Para definir o compilador, edite o arquivo **Makefile** e modifique a primeira linha. O compilador padrão (*default*) é **g77**.

Extração de arquivos e compilação: Digite as seguintes linhas de comando:

```
> gunzip MyGa_1.0.tar.gz
> tar xvf MyGa_1.0.tar
> cd MyGa_1.0
> make
```

3. Utilização

A otimização no MyGa é feita ***minimizando-se*** uma função *f(x)* calculada em **func.f** com **x** um vetor de dimensão **dimx** definido em **parameter.inc**.

3.1 Exemplos

O arquivo **func.f** fornecido contém 5 exemplos de funções testes. O primeiro exemplo é o da função esfera com dimensão *dimx* = 10. Para testar este exemplo digite:

```
> make
> Myga
```

A evolução da melhor solução ao longo das gerações é ao mesmo tempo exibida na tela e salva nos arquivos **bestfx.dat** (o valor da função) e **bestx.dat**. O arquivo **resultat.dat** contém o resultado final obtido. Para os testes com outros exemplos é preciso editar o arquivo **func.f** e outro exemplo é escolhido mudando-se a linha **goto 10** para **goto *num*** onde *num* $\in$ {10, 20, 30, 40, 50}. Edite em seguida o arquivo **parameter.inc** para escolher a dimensão **dimx** em função do caso escolhido. Você pode deixar a dimensão inalterada, salvo para o caso *num* = 40 onde a dimensão deve ser igual a 2. Então edite o arquivo **param.dat** mudando as duas linhas:

```
xmin=10*-1.D1,
xmax=10*1.D1,
```

para fornecer os limites com os quais se efetuará a busca. Troque 10 pelo valor de **dimx** nas duas linhas e troque -1.D1 e 1.D1 respectivamente para o limite inferior e superior dos intervalos de busca. Recompile e reinicie o programa.
De maneira geral para se efetuar uma busca em:

$$E = [xmin_1, xmax_1] \text{ x } \cdots \text{ x } [xmin_{dimx}, xmax_{dimx}]$$

os limites devem ser escritos da seguinte maneira:

xmin= $xmin_1, \cdots, xmin_{dimx},$
xmax= $xmax_1, \cdots, xmax_{dimx},$

3.2 Utilização simplificada

Para utilizar rapidamente MyGa para minimizar uma outra função proceda da seguinte maneira:

1. Mude no arquivo **parameter.inc** a dimensão do espaço de busca **dimx**. O valor deve ser o número de parâmetros a serem otimizados.
2. Escreva no arquivo **func.f** sua função a minimizar. Esta função deve receber um vetor **x** de dimensão **dimx** e retornar o valor da função **fx**.
   Se sua função corresponde a um programa que você utiliza com suas próprias notações e nomes de variáveis, o melhor é "guardar" suas variáveis a serem otimizadas no vetor x. Assim, você mantém os nomes de suas variáveis e a rotina **func.f** só "vê" o vetor **x** e o valor **fx**. Da mesma maneira, se você tem que transferir parâmetros para a sua função ou fazer inicializações, você deve fazer estas operações sem modificar a estrutura da rotina **func.f**.

3. Mude no arquivo **param.dat** os limites do espaço de busca como indicado na seção 3.1.

4. Compile com **make** e execute o programa.

3.3 Utilização avançada

Depois de realizadas as etapas da seção 3.2, essa seção detalha os arquivos **parameter.inc** e **param.dat**.

3.3.1 O arquivo parameter.inc

Nesse arquivo há 4 parâmetros a serem determinados:

● **popsize**: o tamanho da população de cada geração. Este programa apresentou melhores performances com um tamanho igual a 10.
● **dimx**: a dimensão do espaço de busca.
● **genemax**: o número máximo de gerações antes que o programa seja finalizado.
● **genemaxmut**: é um parâmetro que determina o decrescimento da força de mutação do tipo **nonun2** definida na seção 3.3.2. Este parâmetro dever ser **genemaxmut≥genemax**. Para um decréscimo mais lento da força de mutação experimente **genemaxmut=1.5* genemax**, por exemplo.

3.3.2 O arquivo param.dat

● **Exibição na tela**: o parâmetro **affichecran** pode assumir os valores 0, 1, 2 e 3. Para cada nível informações suplementares são adicionadas.

- 0 : nenhuma exibição na tela.
- 1 : exibição da última geração e da melhor solução.
- 2 : exibe-se o valor da melhor solução encontrada em cada geração.
- 3 : a geração inicial é exibida.

● **Salvar**: o parâmetro **savefich** pode assumir os valores 0, 1 e 2.

- 0: nenhuma gravação. Essa não é uma alternativa interessante se você, além disso, utilizar affichecran=0!
- 1: a melhor solução é salva (arquivo **resultat.dat**) e o estado dos cálculos (arquivo **restart.dat**) também é salvo para uma retomada eventual dos cálculos.
- 2: salva a melhor solução a cada geração (arquivo **bestx.dat**) e o valor da função correspondente (arquivo **bestfx.dat**).

● **Testes de parada**: dois tipos de testes de parada são possíveis. Quando você quiser fazer testes de parada o programa pode parar quando a solução (**solx**) é alcançada ou quando o valor mínimo de *f* (**solfx**) é atingindo. No geral o programa pára quando o número de iterações máximo (**restartfin**) é atingido.

- Solução alcançada: coloque o parâmetro **typeconv='solx',** , **solx=$x_1,\cdots,x_{dimx}$,** e **eps=ε,** onde $(x_1,\cdots,x_{dimx})$ são as coordenadas da solução e ε define a precisão.

- Valor mínimo alcançado: coloque o parâmetro **typeconv='solfx',**, **solfx=minimum,** e **eps=ε,** onde *minimum* é o valor do mínimo da função *f*.

- caso geral: coloque o parâmetro **typeconv=' ',** e **restartfin=iterfin,** onde *iterfin* é a geração na qual o programa deve parar e deve ser $iterfin \leq genemax$.
Mais precisamente **restartini=iterdebut,** e **restartfin=iterfin,** definem a geração do começo e a geração do fim do programa. Para um primeiro cálculo deve-se ter **restartini=1,** . Nesse caso a população inicial é gerada aleatoriamente no espaço de busca $E = [xmin_1, xmax_1] \times \cdots \times [xmin_{dimx}, xmax_{dimx}]$.
Para recuperar um antigo cálculo de uma geração $gen_1 < genemax$ utilize **restartini=gen_1,** e **restartfin=iterfin,** com $gen_1 \leq iterfin \leq genemax$. Nesse caso o cálculo será retomado com a população que será relida no arquivo **restart.dat** criado a partir do primeiro cálculo. Nota-se que um cálculo feito em dois tempos não é sempre o mesmo que um cálculo feito em uma única etapa visto que o arquivo **restart.dat** contém unicamente a população a qual o cálculo deve retomar, mas não contém outras informações. Este detalhe não tem muita importância salvo nos casos onde são feitos testes de performance ou onde se busca a reprodução perfeita de um cálculo antigo.

● **Inicialização da população**: A população inicial é gerada aleatoriamente no caso geral. Quando o parâmetro **initpop=m,** com $0<m\leq dimx$, o programa vai ler *m* indivíduos no arquivo **initx.dat** que vão substituir *m* indivíduos da população aleatória. Nesse caso você deve introduzir estes *m* indivíduos no arquivo **initx.dat** com uma coordenada por linha. Isto pode ser útil se você introduzir indivíduos que lhe pareçam próximos da solução. Mas

atenção à convergência prematura que pode ser provocada por um indivíduo muito bom (aquele que você adicionou) em relação aos outros.

● **Espaço de busca**: Como indicado na seção 3.1, o espaço de busca é definido pelos limites **xmin=** $xmin_1, \cdots, xmin_{dimx},$ e **xmax=** $xmax_1, \cdots, xmax_{dimx},$. O programa não leva em conta qualquer outro tipo de restrição. Uma maneira simples de levar em conta outras restrições é de penalizar fortemente uma violação de restrição na função de custo *f*. Não é a melhor maneira de levar em conta as restrições, mas na falta de outra alternativa melhor, é uma maneira que funciona em muitos casos com restrições simples.

● **Cruzamento**: Os parâmetros que correspondem ao operador cruzamento são **pc**, **typec**, **alpha**, **nbmultip** e **tm**.
A probabilidade de se aplicar um cruzamento entre cromossomos é dada por **pc**.
Quatro tipos de cruzamento são possíveis:

- **typec='bary',**: cruzamento baricêntrico com o parâmetro **alpha=α,** definindo os pesos α e (1- α).
- **typec='baryal',**: cruzamento baricêntrico onde o coeficiente α é escolhido aleatorimente no intervalo [0,1].
- **typec='multip',**: cruzamento multipontos com o número de pontos de corte dado por **nbmultip=m,** onde *m* é um inteiro $0<m<dimx$.
- **typec='sbx',**: cruzamento dito SBX com o parâmetro **alpha= α,** definindo a forma da lei de probabilidade utilizada. Tome α = 1 ou 2 por exemplo.

No caso onde os dois pais são muito próximos, a probabilidade de mutação do filho (ver o item sobre a mutação abaixo) é aumentada e torna-se igual à **tm**. O objetivo é de provocar mutação nas duas crianças geradas que se pareçam muito com seus pais.

● **Mutação**: Os parâmetros deste operador são: **pm**, **typem**, **b**, **nbstag** e **kb**.
A probabilidade de aplicar uma mutação a um indivíduo é dada por **pm**.
Três tipos de mutação são possíveis:

- **typem='varnor',**: mutação gaussiana com um desvio tipo σ que decresce em direção a zero com o curso das gerações.
- **typem='nonuni',**: mutação não uniforme (ver [ref. 1, (Seção 1.3.2.4)] e [ref. 3] para a definição dessa mutação) Esta mutação é definida pelo parâmetro **b**. Um pequeno valor de b (~1) favorece a exploração e um valor elevado acelera a busca local.
- **typem='nonun2',**: é uma variação da mutação não uniforme que frequentemente dá os melhores resultados.

Escapar dos mínimos locais: no caso das mutações do tipo **nonuni** e **nonun2**, os parâmetros **nbstag** e **kb** permitem a saída de alguns mínimos locais nos quais o programa pode se encontrar aprisionado. De fato, se nenhuma melhora na melhor solução é conseguida após **nbstag** gerações, o valor do parâmetro **b** torna-se, naquela

geração, **kb*b** com **kb< 1**. Isto tem como efeito o aumento da força de mutação e favorece a exploração no espaço de busca.

● **Busca local**: definida pelos parâmetros **RafLoc** e **nbtot**. Este procedimento é pouco eficaz e resulta frequentemente numa convergência prematura. Para evitar, deixe **RafLoc='N0rafloc',** .

● **Seleção**: dois tipos de seleção são possíveis.
- **types='roulet',**: seleção por roleta.
- **types='rs',**: seleção por roleta estocástica.

● **Elitismo**: ativado se **elit=1,** e desativado se **elit=0,**. O elitismo consiste em assegurar que não haja decréscimo da melhor solução a cada geração. Na prática, se para uma dada geração *n* nenhum indivíduo é melhor que a solução da geração anterior *n-1*, o melhor indivíduo da geração *n-1* é transferido para a geração n.

● **Nichos** (***Niching***): esta técnica permite favorecer a exploração espacial penalizando os indivíduos que se encontram na mesma região do espaço. Os parâmetros são **niche**, **q**, **sigmaniche** e **alphaniche**. Ver [ref. 3] pra os detalhes dessa técnica.

- **niche=1,** : para ativar esta técnica e 0 para desativá-la.
- **q=p,**: para utilizar a norma $\|\cdot\|_p$ no cálculo da distância.
- **sigmaniche=σ,**: deve ser < 1.
- **alphaniche= α,**: para determinar a intensidade do nicho. Com α pequeno (α <1) o nicho é pouco intenso e com α grande o nicho é bastante intenso.

● **Escalonamento**: ver [ref. 1, (Seção 1.3.2.4)] e [ref. 2] para os detalhes. Os parâmetros são **scaling**, **scal**, **scalingvar**, **scalvar1**, **scalvar2** e **scalvarp**. Os dois primeiros parâmetros para o primeiro escalonamento (5 possibilidades) e os quatro últimos para um segundo escalonamento variável (facultativo) em função do tempo (5 possibilidades). O escalonamento que deu os melhores resultados é o *σ-scaling* que é escolhido por **scaling='sigma',**. Os outros escalonamentos possíveis são **exp** (em e^x), **xexp** (em xe^x), **log** (em log(1+x)) e **puiss** (em x^{scal}). Estes mesmos escalonamentos são adaptados para o escalonamento variável.

4. Comentários

● **Inicialização do gerador de números aleatórios**: atualmente o gerador de números aleatórios é inicializado sempre da mesma maneira. Como conseqüência, duas execuções sucessivas do programa produzem o mesmo resultado. Para inicializar o gerador de outra maneira, edite o arquivo **MyGa.f** e mude no começo do programa a linha **irandom=-1** para **irandom=time()**.

Referências

[1] A. Ben Haj Yedder. *Optimisation numérique et Contrôle optimal : (applications en chimie moléculaire)*. PhD thesis, Ecole Nationale des Ponts et Chaussées, 2002.

[2] Z. Michalewicz. *Genetic algorithms + data structure = evolution programs*. Springer, 1999.

[3] Mourad Sefrioui. *Algorithmes Evolutionnaires pour le calcul scientifique. Application la mécanique des fluides et l'électromagnetisme.* PhD thesis, Université Pierre et Marie Curie, 1998.